INTRODUCTION

- **A simple example:**
  - **Job: put on socks and shoes**
  - **Processor: a pair of hands**
  - **Sequential algorithm:**

    put on right sock, right shoe,

    put on left sock, left shoe.

    Need 4 time units
  - **Parallel algorithm:**

    Two processors:

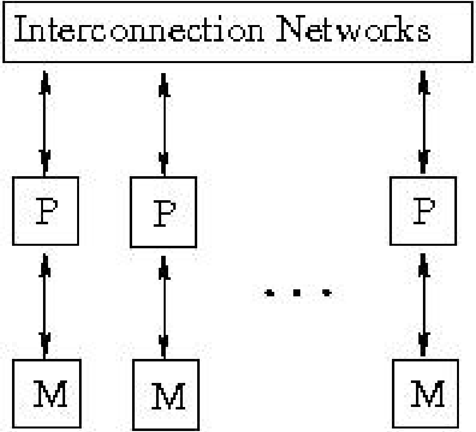
    one for left foot and another for right foot.

    Need 2 time units.

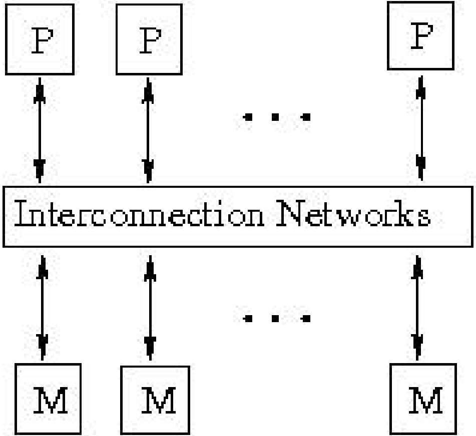
    Question: Can we use four processors to further speed up to, say, 1 time unit?

- **Parallel computer models**
  - **Physical architecture models**
    - **Multiprocessors**
      - **Uniform memory access (UMA), a single shared memory space.**
      - **Nonuniform memory access (NUMA), distributed shared-memory multiprocessors (DSM).**
    - **Multicomputers (distributed memory)**
      - **Hypercube architecture**
      - **Mesh connected architecture**
    - **Networks of workstations (NOW)**

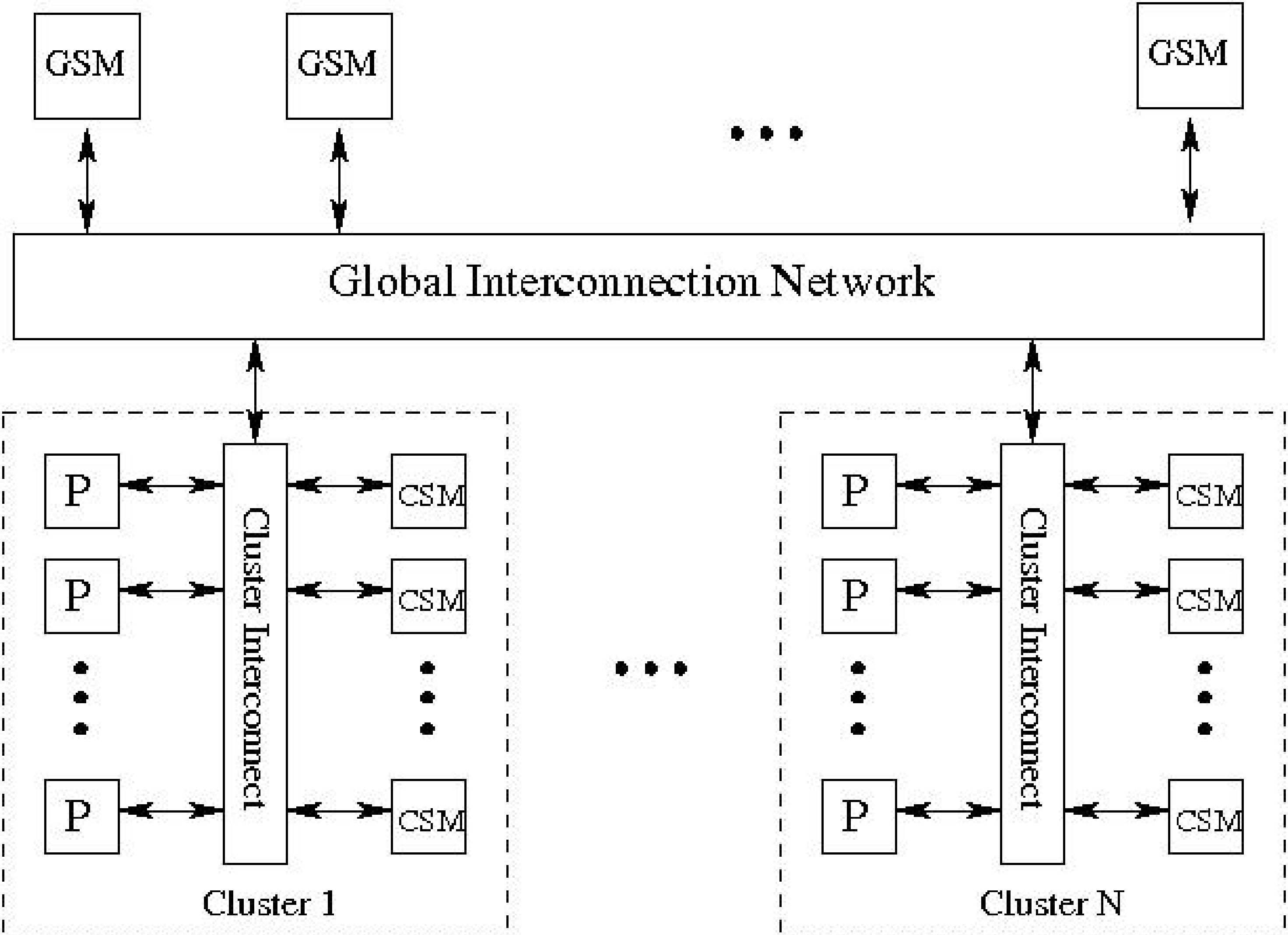
      An inexpensive way to build parallel computers.



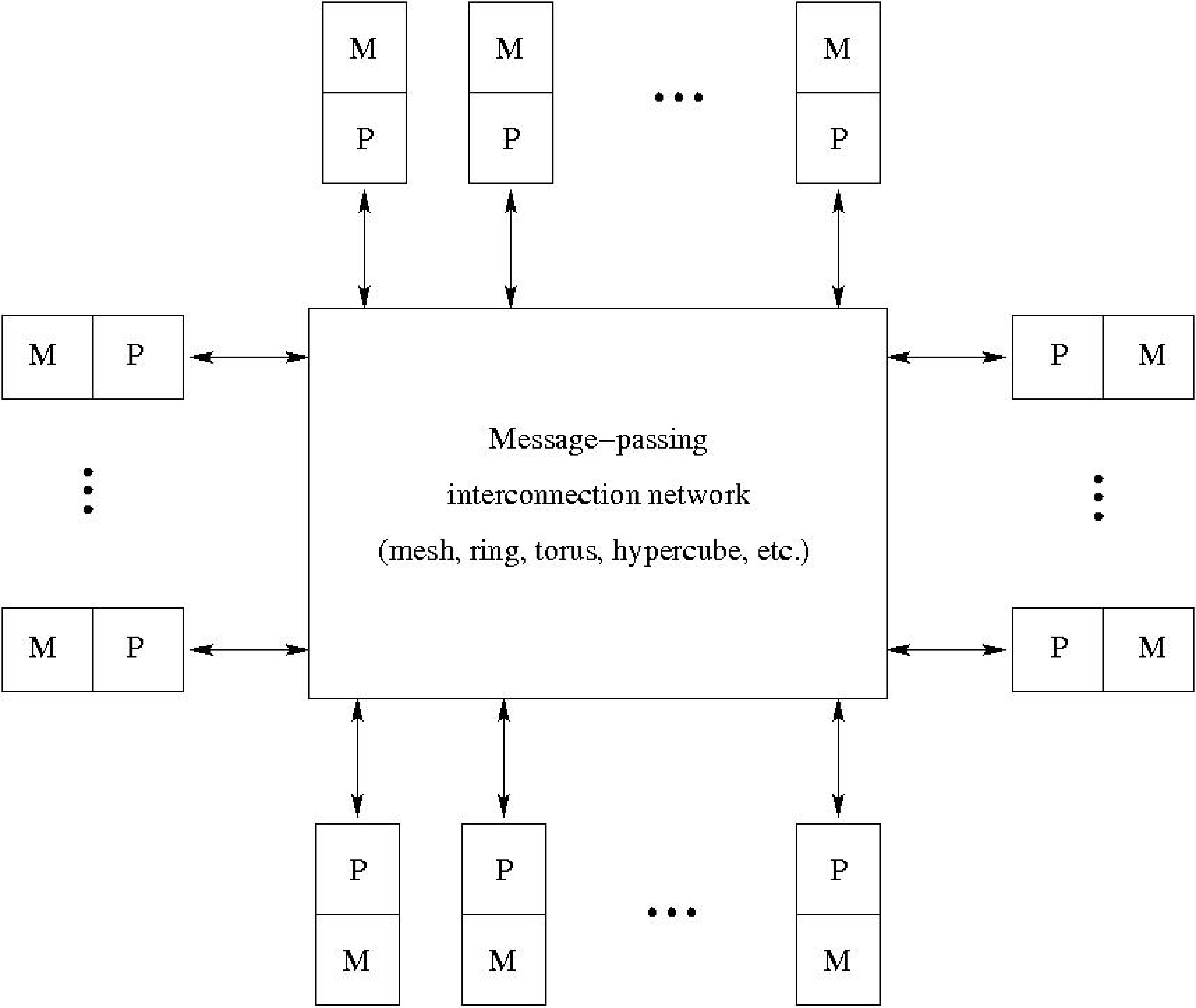
A multicomputer



A UMA shared-memory multiprocessor



NUMA model for multiprocessor system



A message passing multicomputer

– **Theoretical models**

Used to estimate the performance bounds on algorithms.

∗ **Review of time and space complexity**

· **Time complexity: a function of the problem size**

· **Big O notation (worst case complexity):**
a time complexity $g(n)$ **is said to be** $O(f(n))$ **if there exist positive constants** c **and** n_0 **so that** $g(n) \leq cf(n)$ **for all nonnegative values of** $n > n_0$**.**

· **Sequential complexity: the complexity of sequential algorithm**

· **Parallel complexity: the complexity of parallel algorithm**

- **NP-problems**
  - **An algorithm has time complexity** $O(f(n))$ **where** n **is the problem size.**
  - **P-class (polynomial):** $f(n)$ **is a polynomial.**
  - **NP-class (nondeterministic polynomial): polynomial verifiable for a guessed solution, but** $f(n)$ **is exponential.**
- **Examples:**

  P-class: search max in a list: $O(n)$

  NP-class: Traveling salesman problem

  (travel all cities with minimum cost): $O(n^2 2^n)$**.**

$*$ **Parallel complexity**

- **Sequential complexity** $O(w(n))$
- **Parallel complexity of a** p**-processor machine** $O(\frac{w(n)}{p})$**:**
  the algorithm is scalable.
- **Not every problem can achieve this due to data dependence**
- **An example:**
  putting on socks and shoes

$*$ **Parallel random access machine (PRAM). Consists of**

- p **processors** $P_1, \ldots, P_p$
- **Processors are connected to a large shared, random access memory** M**.**
- **Processors have a private or local memory for their own computation, but all communication among them takes place via the shared memory**
- **Each time step has three phases: read phase, computation phase and write phase.**
- **Processors synchronized (write at the same time)**

* **Four subclasses, depending on how concurrent read/write is handled:**
  - **EREW-PRAM: exclusive read exclusive write. Allow only one processor to read or write a memory location**
  - **CREW-PRAM: concurrent read exclusive write. Allow multiple processors to read the same memory location, but not allow concurrent write.**
  - **ERCW-PRAM: exclusive read concurrent write.**
  - **CRCW-PRAM: concurrent read current write.**

∗ **How to resolve the write conflicts**

- **Common: all simultaneous writes store the same value to that memory location**
- **Arbitrary: choose one value ignore others**
- **Minimum: store the value of the processor with the minimum index**
- **Priority: some combination of all values, such as summation or maximum**

∗ **In PRAM model, synchronization and memory access overhead are ignored.**

$\ast$ **Example:**

An algorithm on a PRAM:

Multiplication of two $n \times n$ **matrices in** $O(\log n)$ **time on a PRAM (CREW) with** $n^3/\log n$ **processors.**

$$A \times B = C$$

$$A(i,k), B(k,j), C(i,j,k), \quad 0 \leq i,j,k \leq n-1$$

First assume n^3 **processors:**

$$PE(i,j,k), \quad 0 \leq i,j,k \leq n-1$$

Standard algorithm:

$$C(i,j) = \sum_{k=0}^{n-1} A(i,k) \times B(k,j)$$

We put the final results in $C(i,j,0)$ **for** $0 \leq i,j \leq n-1$**.**

Step 1:

$$C(i,j,k) = A(i,k) \times B(k,j)$$

Step 2:

$$C(i,j,0) = \sum_{k=0}^{n-1} C(i,j,k)$$

Now look at $n^3/\log n$ **processors.**

$$C(i,j,k), \quad 0 \le i,j, \le n-1, \quad 0 \le k \le \frac{n}{\log n} - 1$$

Step 1:

$$C(i,j,0) = \sum_{k=0}^{\log n-1} A(i,k) \times B(k,j)$$

$$C(i,j,1) = \sum_{k=\log n}^{2\log n-1} A(i,k) \times B(k,j)$$

$$\vdots$$

$$C(i,j,n/\log n - 1) = \cdots$$

Step 2:

$$C(i,j,0) = \sum_{k=0}^{n/\log n-1} C(i,j,k)$$

Modify the code: $l \leftarrow n$ **to** $l \leftarrow n/\log n$

Algorithm :

Step 1:

1. Read A(i, k)
2. Read B(k, j)
3. Compute $A(i,k) \times B(k,j)$
4. Store in C(i, j, k)

Step 2:

1. $\ell \leftarrow n$
2. **Repeat**

```
        ℓ ← ℓ/2
        if (k ≤ ℓ) then
            begin
                Read C(i, j, k)
                Read C(i, j, k + ℓ)
                Compute C(i, j, k) + C(i, j, k + ℓ)
                Store in C(i, j, k)
            end
   until (ℓ = 1)
```

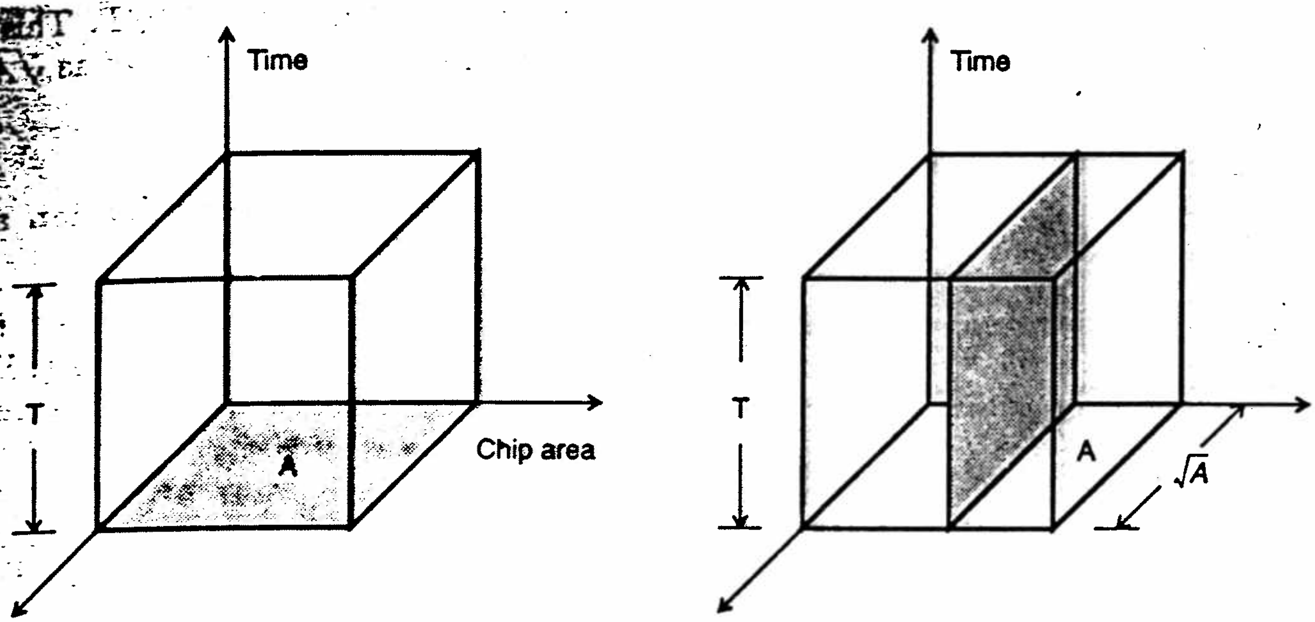
∗ VLSI complexity model (AT^2 model)

- **Set limits on memory, I/O and communication, for implementing parallel algorithms with VLSI chips.**
- **A: chip area (chip complexity)**
- **T: time for completing a given computation**
- **s: problem size**
- **There exists a lower bound $f(s)$ such that**

$$A \times T^2 \geq O(f(s))$$

- **Memory requirement sets a lower bound on chip area A**
- **Information flows through the chip for a period of time T.**

- **AT: the amount of information flowing through the chip during time T. The number of input bits cannot exceed the volume AT.**
- **Bisection $\sqrt{A}T$ (usually use AT^2): maximum information exchange between the two halves of the chip during time T.**



(a) Memory-limited bound on chip area A and I/O-limited bound on chip history represented by the volume AT

(b) Communication-limited bound on the bisection $\sqrt{A}T$

Figure 1.15 The AT^2 complexity model of two-dimensional VLSI chips.

- **Example:**

  Matrix multiplication.

  $n \times n$ **matrices,** $C = A \times B$

  2-D mesh architecture, n^2 **PE's**

  broadcast bus for inter-PE communication

  chip area complexity: $A = O(n^2)$

  time complexity $T = O(n)$

$$AT^2 = O(n^2) \cdot (O(n))^2 = O(n^4)$$

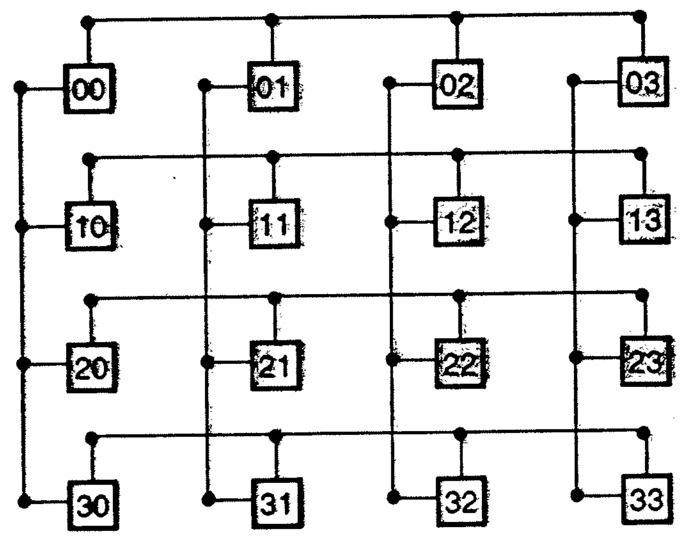


Figure 1.16 A 4 × 4 mesh of processing elements (PEs) with broadcast bu each row and on each column. (Courtesy of Prasanna Kumar and Ra; dra; reprinted from *Journal of Parallel and Distributed Computing*, Apri

```
     Do 50 for 0 ≤ k ≤ n − 1
         Doall 20 for 0 ≤ i ≤ n − 1
20           PE(i,k) broadcasts A(i,k) along its row bus
         Doall 30 for 0 ≤ j ≤ n − 1
30           PE(k,j) broadcasts B(k,j) along its column bus
                 /PE(i,j) now has A(i,k) and B(k,j), 0 ≤ i,j ≤ n − 1/
         Doall 40 for 0 ≤ i,j ≤ n − 1
40           PE(i,j) computes C(i,j) ← C(i,j) + A(i,k) × B(k,j)
50   Continue
```

- **How to solve a typical computation task sorting using different types of computation models.**
  - **Problem description:**

    A sequence

    $$S = \{s_1, s_2, \ldots, s_n\}$$

    A linear order $<$ **is defined on** S**.**

    Find a new sequence

    $$S' = \{s'_1, s'_2, \ldots, s'_n\}$$

    such that $s'_i < s'_{i+1}$ **for** $i = 1, 2, \ldots, n-1$**.**

– Sequential algorithm.

- $*$ **Lower bound:** $\Omega(n \log n)$
- $*$ **Mergesort (optimal)**

$$\textbf{Time } T(n) = O(n \log n)$$

– **Parallel algorithm on CRCW model.**

- **Write conflict: storing the sum of all values being written.**
- **Sorting by enumeration:**

  n^2 **processors.**

  Two lists in shared memory:

  S **stores** $s_1, s_2, \ldots, s_n$ **and** C **stores** $c_1, c_2, \ldots, c_n$

  c_i **is the number of of elements in** S **smaller than** s_i**.**

  If $s_i = s_j$ **and** $i > j$ **then** $s_i > s_j$ **in the sorted list.**
- **Each** $p(i, j)$ **compares** s_i **and** s_j **and stores** s_i **in position** $1 + c_i$ **of** S**.**
- **Time** $T(n) = O(1)$
- **Processors:** $P(n) = n^2$
- **Cost:** $C(n) = T(n)P(n) = O(n^2)$
- **This algorithm is not optimal.**

  If $c(n) = O(n \log n)$ **optimal.**

Procedure CRCW sort(S)

Step 1: for $i = 1$ **to** n **doall**

for $j = 1$ **to** n **doall**

if $(s_i > s_j)$ **or** $(s_i = s_j$ **and** $i > j)$

then $p(i, j)$ **writes 1 in** c_i

else $p(i, j)$ **writes 0 in** c_i

end if

end for

end for

Step 2: for $i = 1$ **to** n **doall**

$P(i, 1)$ **stores** s_i **in position** $1 + c_i$ **of** S

end for

– Parallel algorithm on CREW model.

Divide S into p subsets and one processor sorts a subset.

$$S = S_1 \cup S_2 \cup \cdots \cup S_p$$

$$\begin{aligned} T(n) &= O(\log^2 n) \\ P(n) &= O(n/\log n) \\ C(n) &= O(n \log n) \end{aligned}$$

Optimal algorithm.

– **A special purpose parallel architecture designed for sorting (hardware sorter)**

Specialized processors $+$ custom-designed inter-connection networks

Odd-even sorting network

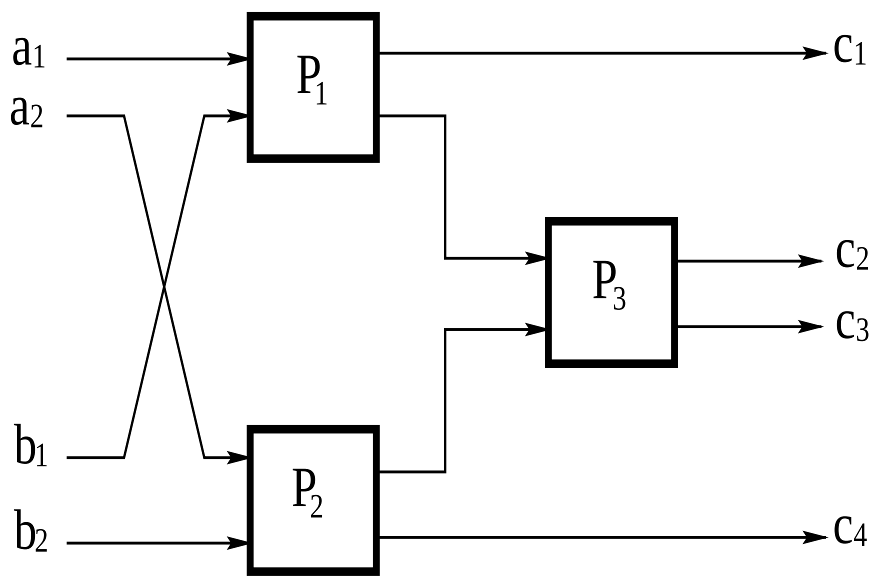
Very simple processor: 2×2 comparator

Basic idea: merge sort

(n, n) **merging network: merges two length-n sorted lists into one length $2n$ sorted list.**

- $*$ $(1, 1)$ **merging network** $= 2 \times 2$ **comparator**
- $*$ $(2, 2)$ **merging network**

$$A = \{a_1, a_2\}, \quad B = \{b_1, b_2\}$$



$$a_1 \leq a_2, \quad b_1 \leq b_2$$

$$\min\{a_1, b_1\} = \min\{a_1, a_2, b_1, b_2\} = c_1$$

$$\max\{a_2, b_2\} = \max\{a_1, a_2, b_1, b_2\} = c_4$$

One more comparator to compare c_2 and c_3.

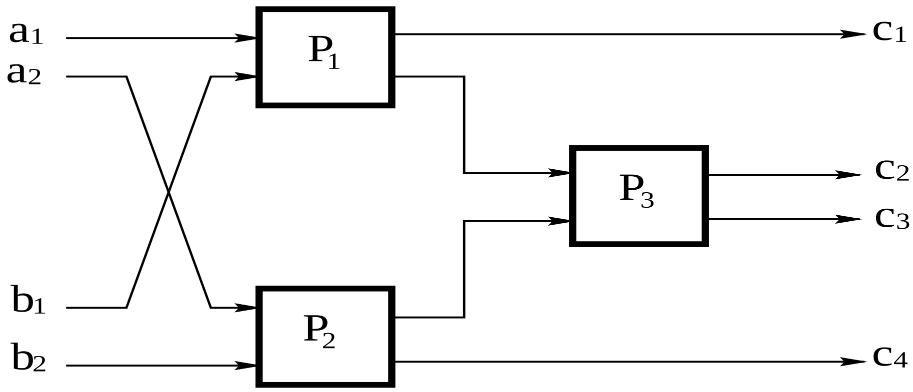
$*$ (n, n) **merging network(n is a power of 2):**

Recursive construction using two $(n/2, n/2)$ **merging networks**

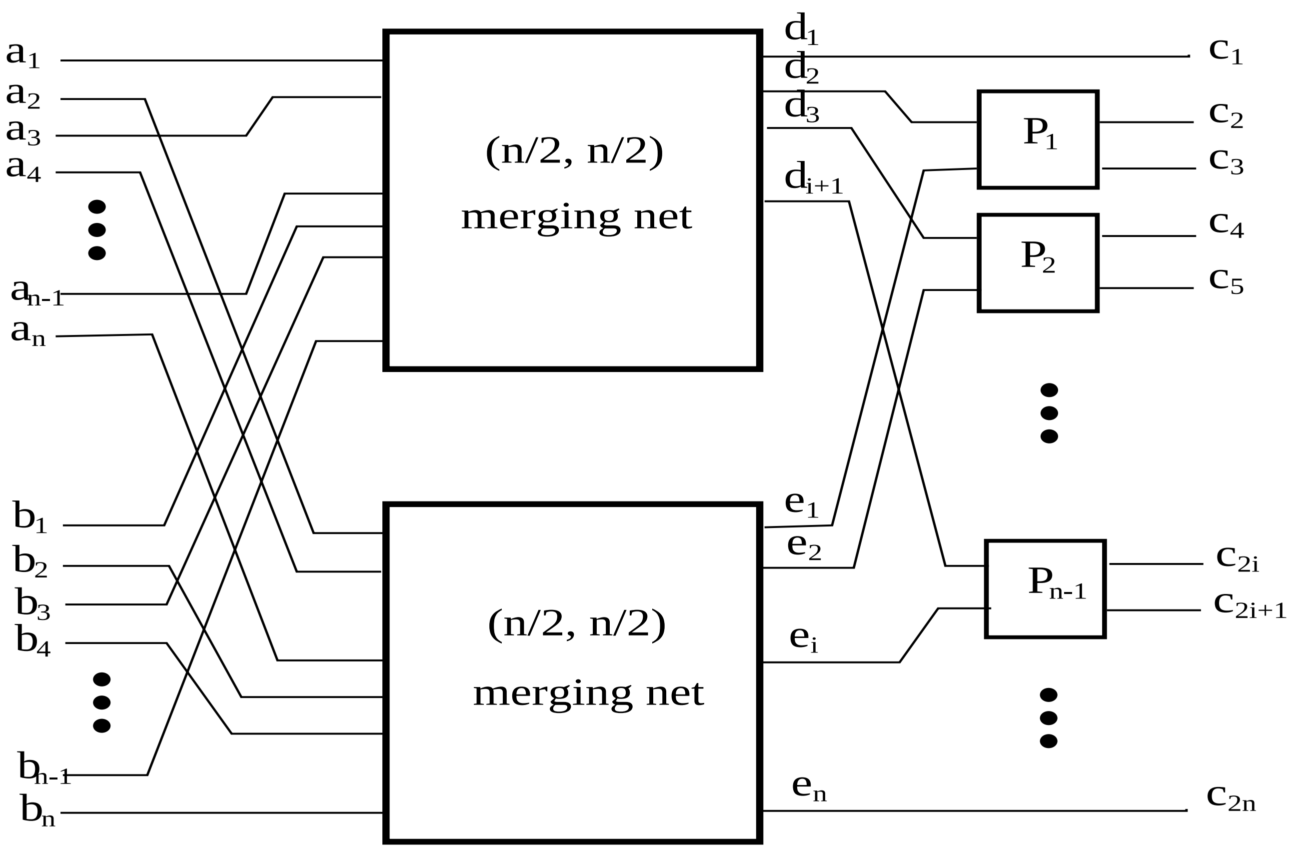
$a_1, a_3, \ldots, a_{n-1}, b_1, b_3, \ldots, b_{n-1}$ **connected to the first merger**

$a_2, a_4, \ldots, a_n, b_2, b_4, \ldots, b_n$ **connected to the second merger**

Additional $n - 1$ **comparators**



a1
a2
b1
b2
P1
P2
P3
c1
c2
c3
c4



a1
a2
a3
a4
an-1
an
b1
b2
b3
b4
bn-1
bn
(n/2, n/2)
merging net
(n/2, n/2)
merging net
d1
d2
d3
di+1
e1
e2
ei
en
P1
P2
Pn-1
c1
c2
c3
c4
c5
c2i
c2i+1
c2n

Proof of correctness.

Note that subsequences $a_1, a_3, \ldots, a_{n-1}$ **and** $b_1, b_3, \ldots, b_{n-1}$ **are sorted, and we have**

$$d_1 \leq d_2 \leq \cdots \leq d_n$$

$$e_1 \leq e_2 \leq \cdots \leq e_n$$

d_1 **is the min of all elements** $\Rightarrow d_1 = c_1$
e_n **is the max of all elements** $\Rightarrow e_n = c_{2n}$

Now, we need to prove:

$$c_{2i} = \min\{d_{i+1}, e_i\}$$

$$c_{2i+1} = \max\{d_{i+1}, e_i\}$$

Consider sequence $\{d_1, d_2, \ldots, d_{i+1}\}$**:**

$$\{d_1, d_2, \ldots, d_{i+1}\} \subseteq \{a_1, a_3, \ldots, a_{n-1}, b_1, b_3, \ldots, b_{n-1}\}$$

Suppose k **elements of** $\{d_1, d_2, \ldots, d_{i+1}\}$ **are in** $\{a_1, a_3, \ldots, a_{n-1}\}$

They must be the first k elements

$$\{a_1, a_3, \ldots, a_{2k-1}\}$$

Then $i+1-k$ elements in $\{b_1, b_3, \ldots, b_{n-1}\}$. These elements must be the first $(i+1-k)$ elements

$$\{b_1, b_3, \ldots, b_{2(i+1-k)-1}\}$$

Look at the largest element d_{i+1},

$$d_{i+1} \geq \{a_1, a_3, \ldots, a_{2k-1}\}$$

Plug in

$$\{a_2, a_4, \ldots, a_{2k-2}\}$$

d_{i+1} **is greater than $2k-1$ a_i's**

Similarly, d_{i+1} is greater than $2(i+1-k)-1$ b_i's

$$2k-1+2(i+1-k)-1 = 2i$$

Then we have

$$d_{i+1} \geq c_{2i}$$

Similarly, consider $\{e_1, e_2, \ldots, e_i\}$**.**

k **of** $\{e_1, e_2, \ldots, e_i\}$ **are in** $\{a_2, a_4, \ldots, a_n\}$**.**

$i - k$ **of** $\{e_1, e_2, \ldots, e_i\}$ **are in** $\{b_2, b_4, \ldots, b_n\}$**.**

e_i **is greater than** $2k$ a_i**'s, and** e_i **is greater than** $2(i-k)$ b_i**'s.**

So

$$e_i \geq c_{2i}$$

We have

$$\begin{aligned} d_{i+1} &\geq c_{2i} \\ e_i &\geq c_{2i} \end{aligned}$$

for $i = 1, 2, \ldots, n-1$**.**

Now let $i = n - 1$**, we have**

$$\begin{aligned} d_n &\geq c_{2n-2} \\ e_{n-1} &\geq c_{2n-2} \end{aligned}$$

Since $e_n = c_{2n}$**,**

$$\{d_n, e_{n-1}\} = \{c_{2n-2}, c_{2n-1}\}$$

Then

$$c_{2n-2} \;=\; \min\{d_n, e_{n-1}\}$$
$$c_{2n-1} \;=\; \max\{d_n, e_{n-1}\}$$

For $i = n-2$**,**

$$d_{n-1} \;\geq\; c_{2n-4}$$
$$e_{n-2} \;\geq\; c_{2n-4}$$

$$\{d_{n-1}, e_{n-2}\} = \{c_{2n-4}, c_{2n-3}\}$$

Then

$$c_{2n-4} \;=\; \min\{d_{n-1}, e_{n-2}\}$$
$$c_{2n-3} \;=\; \max\{d_{n-1}, e_{n-2}\}$$

Analysis for merger:

– **Time:**

$T(2) = 1$, $T(2n) = T(n) + 1$

$T(2n) = 1 + \log n$

– **Processors:**

$P(2) = 1$

$P(2n) = 2P(n) + (n - 1)$

$P(2n) = 1 + n \log n$.

– **Cost:**

$C(2n) = P(2n) \times T(2n) = O(n \log^2 n)$

Not optimal ($O(n)$ is optimal).

Back to odd-even sorting network:

- **Time:**

  $T(n) = T(n/2) + (1 + \log(n/2)) = T(n/2) + \log n = O(\log^2 n)$

- **Processors:**

  $P(n) = 2P(n/2) + 1 + (n/2)\log(n/2) = O(n \log^2 n)$

- **Cost:**

  $C(n) = P(n) \times T(n) = O(n \log^4 n)$

Summary for sorting

- **Odd-even sorting network**
  - $T(n) = O(\log^2 n)$
  - $P(n) = O(n \log^2 n)$
  - $C(n) = O(n \log^4 n)$

  Not optimal, but a practical network.
- **Sequential algorithm**
  - $T(n) = O(n \log n)$
  - $P(n) = O(1)$
  - $C(n) = O(n \log n)$

  Optimal.
- **The best parallel algorithm: AKS sorting network (CREW model)**
  - $T(n) = O(\log n)$
  - $P(n) = O(n)$
  - $C(n) = O(n \log n)$

  Optimal, but very large hidden constant, complex.